第7章 文件系统

本章介绍Linux内核是如何维护它支持的文件系统中的文件的，我们先介绍 VFS(Virtual File System，虚拟文件系统)，再解释一下Linux内核的真实文件系统是如何得到支持的。

Linux的一个最重要特点就是它支持许多不同的文件系统。这使 Linux非常灵活，能够与许多其他的操作系统共存。在写这本书的时候， Linux共支持15种文件系统： ext、 ext2、xia、 minix、 umsdos、 msdos 、vfat、 proc、 smb、 ncp、 iso9660、 sysv、 hpfs、 affs 和ufs。无疑随着时间的推移， Linux支持的文件系统数还会增加。

Linux像UNIX一样，系统可用的独立文件系统不是通过设备标识来访问的，而是把它们链接到一个单独的树形层次结构中。该树形层次结构把文件系统表示成一个整个的独立实体。Linux以装配的形式把每个新的文件系统加入到这个单独的文件系统树中。无论什么类型的文件系统，都被装配到某个目录上，由被装配的文件系统的文件覆盖该目录原有的内容。该个目录被称为装配目录或装配点。在文件系统卸载时，装配目录中原有的文件才会显露出来。

在硬盘初始化时，硬盘上的分区结构把物理硬盘化分成若干个逻辑分区。每个分区可以包含一个像EXT2那样的一个独立的文件系统。文件系统用目录将文件按照逻辑层次结构组织起来。目录是记录在物理设备块中的软链接信息。包含文件系统的设备被称为块设备。

IDE硬盘分区/dev/hda1 —— 系统中第一个IDE硬盘驱动器的第一个分区，就是一个块设备。Linux文件系统把这些块设备当作简单的线性块的集合，它不知道也不在意下层物理硬盘的实际结构。每个块设备驱动程序的任务就是把系统读该设备上某一块的请求映射成对本设备有意义的术语，如该块所在的磁道、扇区或柱面号。无论文件系统存在在何种设备上，它都可以按相同的方式进行操作。而且在使用文件系统时，由不同的硬件控制器控制的不同物理介质上的不同文件系统对系统用户是透明的。文件系统既可能在本地系统上的，也可能是通过网络链接装配的远程文件系统。请看下面根文件系统位于 SCSI硬盘上的Linux系统示例：

```
A       E       boot    etc     lib     opt     tmp     usr
C       F       cdrom   fd      proc    root    var     sbin
D       bin     dev     home    mnt     lost+found
```

对文件进行操作的用户和程序都不需要知道 /C实际上是系统中第一个IDE硬盘上的一个装配了的VFAT文件系统，而/E是指从IDE控制器的主IDE硬盘。即使第一个IDE控制器是PCI控制器，而第二个是同时还控制 IDE 接口的CDROM的ISA控制器，这也不会给系统造成任何不便。我可以使用 PPP网络协议和 modem拨号到我所在公司的局域网上。这时，可以把 Alpha AXP Linux系统中的文件系统远程装配到 /mnt/remote目录上。

文件是数据的集合，如：本章的源文本文件就是一个叫做 filesystems.tex的ASCII码的文件。一个文件系统不仅包括该文件系统中所有文件的数据，还包括文件系统的结构信息。它记录下所有Linux用户和进程当作文件看待的信息、目录软链接信息以及文件保护信息等等。而且文件系统必需保证这些信息的安全性，因为操作系统的基本完整性就取决于它的文件系统。没有人会使用一个随机的丢失数据和文件的操作系统。

Linux支持的第一个文件系统是MINIX文件系统，它对用户有很多限制并且性能比较差。MINIX文件系统的文件名不能超过14个字符，最大文件长度是64M字节。初看起来64M字节好像足够大了，但现代的数据库系统需要更大的文件长度。第一个专门为Linux设计的文件系统是EXT(扩展文件系统)，在1992年4月设计完成并为Linux解决了大量的问题。但它在性能上仍有所欠缺。所以在1993年设计了第二个扩展文件系统——EXT2。本章主要介绍的就是这个文件系统。

在EXT文件系统加入到Linux中时，Linux系统发生了一个重大的发展。真实文件系统从操作系统中分离出来，而由一个接口层提供的真实文件系统的系统服务被称为虚拟文件系统(VFS)。VFS使得Linux可以支持许多种不同的文件系统，而这些文件系统都向VFS提供相同的软件接口。由于所有的Linux文件系统的细节都是由软件进行转换的，所有对Linux系统的其余部分和在系统中运行的程序来说这些文件系统是完全相同的。Linux的虚拟文件系统层使得你可以同时透明地装配很多不同的文件系统。

实现Linux虚拟文件系统要使得它对文件的访问要尽可能地快、尽可能地高效，而且一定要确保文件和数据的正确性。这两个要求彼此是不对称的。在每个文件系统被装配使用后，Linux的VFS会在内存中缓存来自于这些文件系统的信息。因此由于对文件或目录的创建、写、删除操作而改变了Linux缓存中的数据时，对文件系统的更新操作要格外小心。如果你能在运行的内核中看到文件系统的数据结构，就会看到那些文件系统读/写的数据块。代表被访问的文件和目录的数据结构可能被创建或删除，而设备驱动程序不停地工作，读取数据、保存数据。这些缓存中最重要的一个是缓冲区缓存，它把独立文件系统访问下层块设备的方法集成起来。每个被访问的块都被放到缓冲区缓存中，并根据它们的状态放在相应的队列中。缓冲区缓存不仅缓存数据缓冲区，它还有助于管理块设备驱动程序的异步接口。

7.1 第二个扩展文件系统EXT2

EXT2是为Linux设计的一个可扩展的高性能文件系统。它是目前为止Linux中最成功的文件系统，也是当前商业销售的Linux的基础。EXT2与其他的许多文件系统一样都是建立在如下前提的基础上：文件中所有数据都记录在数据块中而且这些数据块的长度都是相同的。但由于在创建EXT2文件系统时可以设定块的大小，所以不同的EXT2文件系统块的长度可能不同。文件长度与块长度的整数倍对齐。如果块的长度是1024字节，那么一个1025字节的文件会占用两个1024字节块。不幸的是这表示平均起来每个文件就要浪费半个块。在计算机领域，你通常要在CPU利用率和存储空间磁盘空间利用率之间作出折衷。这时Linux像大多数的操作系统一样，用降低磁盘利用率的办法来降低CPU的工作负荷。文件系统中并不是所有的块都有数据，有些块是用来记录描述文件系统结构信息的。EXT2文件系统通过用inode数据结构来表示系统中每个文件的方法来定义文件系统中的拓扑结构。inode结构包括文件占用了哪些数据块、文件访问权限、文件的更改时间、文件类型等信息。EXT2文件系统的每个文件都由一个inode来表示，而每个inode节点在系统中都有一个唯一标识号。文件系统的所有inode节点都被记录在inode表中。EXT2目录只是一种特殊的文件，它包含指向目录中所有对象的inode节点的指针。

图1-7-1给出了占用块设备一组连续块的EXT2文件系统的结构。从文件系统的角度来说，块设备只是一组可读写的块。文件系统不必关注块是在哪个物理介质上，因为这是设备驱动

程序的工作。

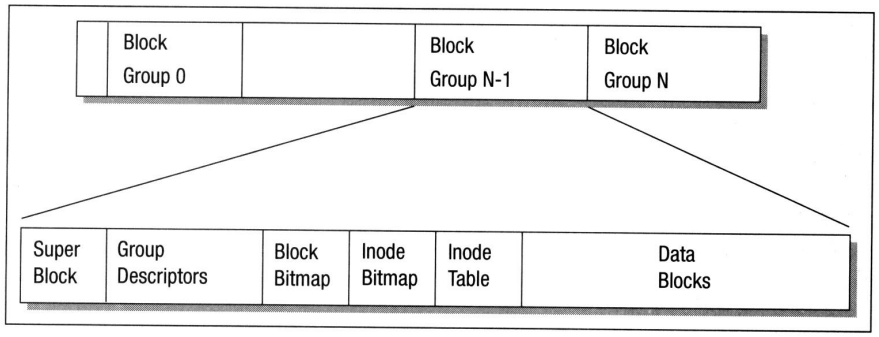

图1-7-1 EXT2文件系统的物理结构

文件系统无论是从块设备上读信息还是数据，它都会向设备驱动程序请求读取块长度的整数倍。EXT2文件系统把它所在的逻辑分区化分成块组。每组除了复制了对文件系统完整性至关重要的信息以外，还包括以数据块形式存放的物理文件、目录等信息。这种备份是必须的，因为一旦系统发生严重灾难，文件系统需要这些信息做恢复工作。下面对每个块组中的内容作详细介绍。

7.1.1 EXT2系统的inode节点

在EXT2文件系统中，inode节点是系统的基本单元。文件系统中的每个文件或目录都由一个inode节点来表示。每个块组中的所有 inode节点都被记录在 inode节点表中，系统使用位图来跟踪inode节点的分配情况。图1-7-2给出了EXT2系统的inode节点的格式。在inode所有域中，有下列几个域比较重要：

- 模式　它包含两部分信息：该inode节点代表哪种文件系统的对象、用户对它的访问许可。在EXT2系统中，一个 inode 节点可以代表文件、目录、符号链接、块设备、字符设备或FIFO管道。
- 拥有者信息　该文件或目录的拥有者的用户标识和组标识。文件系统可以根据它分配正确的访问权限。
- 长度　以字节为单位来表示的文件长度。
- 时戳　该inode节点的创建时间和它最后一次修改的时间。
- 数据块　该域包含指向本 inode节点所代表的文件或目录数据块的指针。前12个指针是直接指向物理块的指针。最后3个指针分别指向 1级到3级的索引。例如2级索引块指针指向一个指针块，而该指针块中的每个指针又指向

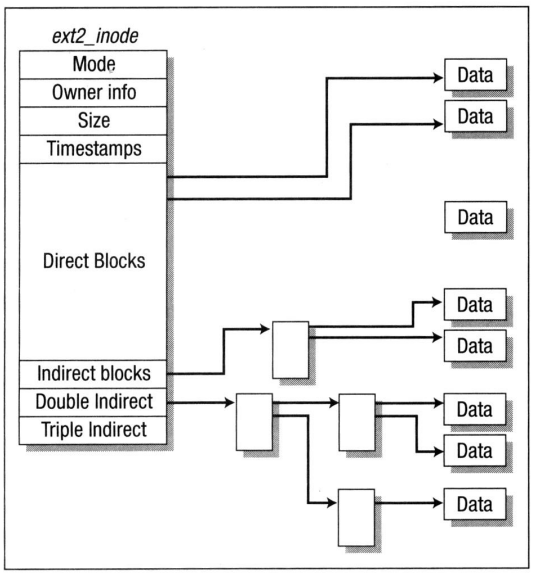

图1-7-2 EXT2的inode结点

一个直接指向数据块的指针块。这种方式使得对长度小于或等于 12个数据块文件的访问比大文件快。

EXT2系统的inode节点还可以表示特殊设备文件。它们不是实际的文件，只是程序用于访问对应的设备的句柄(handle)。在/dev目录下的所有设备文件允许程序通过它们访问 Linux的设备。例如：mount 程序用要装配的设备文件作为输入参数。

7.1.2 EXT2系统的超级块

超级块(Superblock)中包含有对本文件系统的块长度和结构的描述信息。文件系统的管理程序使用超级块中的信息来管理、维护整个文件系统。通常在装配文件系统时，系统只会读取块组0的超级块，但文件系统的每个块组中都有超级块的副本以防止文件系统崩溃。超级块中包含以下一些比较重要的域：

- MagicNumber(魔数) 该域是装配软件用于检查本超级块是否为 EXT2文件系统的超级块的。对当前版本的EXT2系统，该域的值是0xEF53。
- Revision Level 主、次Revision 域允许装配程序检测本文件系统是否支持某些只有在文件系统的某些修正版中才支持的特殊特征。超级块中还有兼容特征域。它可以协助装配程序检测哪些新特征可以在这个文件系统上安全使用。
- 装配记数和最大装配记数 这两个域一起帮助系统决定是否作完全的文件系统检查。每次装配文件系统时，装配记数值都会增 1。如果它等于最大装配记数了，系统会显示“达到最大装配记数，推荐运行 ezfsck”的警告信息。
- 块组号 存放本超级块副本的块组号。
- 块长度 以字节为单位表示的本文件系统块的长度，如 1024字节。
- 每组的块数 每组中块的数目。像块的长度一样，它是在文件系统建立时固定下来的。
- 自由块数 文件系统中空闲块的数目。
- 自由inode数 文件系统中空闲inode节点数。
- 第一个inode节点 本域中存放文件系统中第一个 inode节点的节点号。EXT2根文件系统的第一个inode节点是‘/’目录的目录项。

7.1.3 EXT2系统的组描述符

每个块组都有一个描述它的数据结构。像超级块一样，所有块组的所有组描述符在每个块组中都有副本，以防止文件系统崩溃。每个组描述符包括如下信息：

- 块位图 存放该块组(Block Group)块分配位图的块号，该域在系统分配、释放块时使用。
- Inode位图 本块组的inode分配位图的块号。该域在系统分配、释放 inode节点时使用。
- inode表 本块组的inode节点表的起始块块号。每个 inode节点由EXT2系统的inode数据结构来表示。

自由块记数、自由inode记数、使用的目录数

组描述符在块组中一个接一个存放，它们一起组成了组描述符表。每个块组在超级块的副本之后包含了整个组描述符表。 EXT2文件系统实际上只使用第一个副本 (在块组0中)。其他的副本像超级块的副本一样，用于防止主副本损坏。

7.1.4 EXT2系统的目录

在EXT2文件系统中，目录是一种特殊的文件。它用于创建、保持对文件系统中文件的访问路径。图1-7-3给出了内存中一个目录项的结构图。目录文件是一组目录项的列表，每个目录项包含下列信息：

- inode节点号　该目录项的inode节点号。它是块组的inode表中记录的inode数组的索引值。在图1-7-3中，一个叫“ file ”的文件的目录项中包含有值为i1的inode索引值。

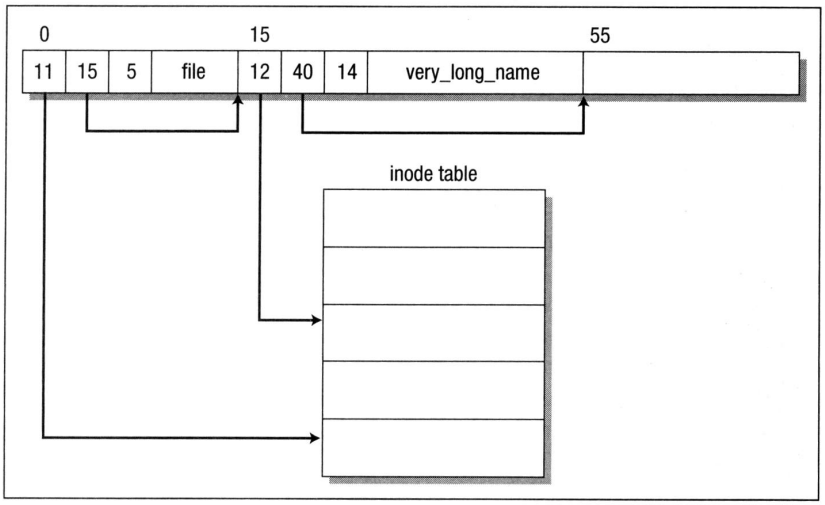

图1-7-3　EXT2系统的目录

- 名字长度　以字节为单位来表示本目录项的长度。
- 名字　本目录项的名字

每个目录的前两项总是标准的“ . ”和“ .. ”目录项，它们分别表示本目录和父目录。

7.1.5 在EXT2文件系统中查找文件

Linux的文件名与UNIX文件名有相同的格式，它是用“ / ”分隔的一组目录名，并且最后以文件名来结尾。文件名的例子是“ /home rusling/.cshrc ”，其中/home 和/rusling是目录名而文件名是.cshrc。像UNIX系统一样，Linux不注意文件名的格式问题。文件名可以任意长，可以包含任何可打印的字符，为了在 EXT2文件系统中找到代表该文件的 inode节点，系统按每次一个目录的方法解析文件名，直到最后到达要找的文件。

文件名解析时，我们需要的第一个 inode节点就是文件系统的根inode节点，在文件系统的超级块中有它的节点号。为了读 EXT2系统的inode节点，我们必须要在相应的块组的 inode表中查找该inode节点。例如：如果根inode节点号是42的话，我们需要块组0的inode表中第42个inode节点。根inode节点是EXT2系统的目录，换句话说就是，根 inode节点的模式把它描述成了一个目录，它的数据块中包含着 EXT2系统的目录项。

“ home ”只是众多目录项中的一个，但 home目录项给出了描述 /home目录的inode节点的节点号。为了查找rusling目录项，我们不得不读取 /home目录(通过先读home的inode节点，再读取由它的inode节点记录的所有包含目录项信息的数据块)。在rusling目录项中给出了描述

/home/rusling目录的inode节点的节点号。最后读出代表 /home/rusling目录的inode节点指向的目录项，在其中找出 .cshrc文件的inode节点号。至此我们获得了包含 file文件信息的数据块。

7.1.6 在EXT2文件系统中改变文件的大小

文件系统的一个共同问题是文件碎片。由于记录文件数据的块散布在文件系统的各处，数据块间离的越远对文件执行顺序的数据块访问的效率就越低。 EXT2文件系统试图通过给文件分配的新块要尽量与文件的当前数据块物理上接近或至少与它的当前数据块在同一个块组的办法来克服文件碎片问题。而只有在上面的条件都无法满足时， EXT2文件系统才会在另一个块组为文件分配数据块。

在进程向文件中写数据时，Linux文件系统要查看数据是否超过了文件的最后一个分配块。如果超过了，那么系统会为该文件再分配一个新的数据块。在分配操作完成之前，进程不能处于执行状态。在进程继续执行之前，它必须等待文件系统分配一个新的数据块，并把数据的剩余部分写入到新块中。 EXT2文件系统的块分配例程做的第一件事情就是锁定该文件系统的超级块。分配或释放块会改变超级块的某些域，而 Linux文件系统不允许一个以上进程同时操纵超级块。如果还有一个进程要分配数据块，它必须等待本进程完成。等待超级块的进程被系统挂起，在超级块的当前用户放弃控制权之前无法进入运行状态。对超级块的访问遵循先来先服务的原则，一旦一个进程取得了超级块的控制权，在完成操作之前它就一直保持对超级块的控制。进程锁定超级块之后，它先检查系统中是否有足够的空闲块。如果系统没有足够的空闲块，分配新块的操作失败，进程会放弃对文件系统超级块的控制。

系统中如果有足够的空闲块的话，系统就会为该进程分配一个。如果 EXT2文件系统支持预分配数据块，那么我们可以从预分配的数据块中直接拿一个。预分配的数据块并不是真的存在，它们只是被保存在分配块的位图中，需要分配新块的代表 file文件的VFS inode节点有两个特殊的EXT2文件系统域：prealloc_block和prealloc_count。其中prealloc_block指第一个预分配数据块的块号， prealloc_count指系统中共有多少个预分配数据块。如果系统中没有预分配的数据块或文件系统不允许块预分配机制，那么 EXT2文件系统必须要真正地分配一个新块。EXT2文件系统要先查看该文件最后一个数据块后面的数据块是否空闲。从逻辑上讲，由于这种分配方法使顺序访问更快，所以它是最高效的文件块分配方法。如果该块不是空闲的，则搜索继续进行，系统会在与理想块距离为 64个块的范围内查找一个空闲的数据块。这一块虽然不是最理想的，但至少非常接近，并与该文件的其他块处在同一块组中。

如果满足上面条件的数据块也找不到，那么进程会逐个地查找所有其他的块组，直到它找到了空闲块。块分配程序可以在块组中寻找一簇 8个连续的空闲块。若不能找到 8个连续的块，那么它也可以寻找少一些的连续空闲块。当块预分配进程被启动后，块预分配进程会相应地更新prealloc_block和prealloc_count两个域。

进程找到空闲块后，块分配程序会更新块组中的块位图，并在缓冲区缓存中为它分配一个数据缓冲区。这个数据缓冲区由文件所在文件系统的支持设备标识和新分配块的块号来唯一标识。缓存区中的数据先被清 0，然后文件系统把它标志为“ 脏 ”以表明该缓冲区中的数据还没有被回写到物理硬盘上。最后超级块解锁并被标记为“ 脏 ”来表明超级块被改变了。如果系统中有等待超级块的进程，那么等待队列中的第一个进程会进入运行状态，并获得对超级块的独占控制权。进程的数据被写入到新的数据块中，如果新块又被填满了，那么整个进

程会重复上面的块分配过程，分配下一个新块。

7.2 虚拟文件系统

图1-7-4给出了Linux内核的虚拟文件系统与它的真实文件系统之间的关系。虚拟文件系统要管理在任何时间装配的所有不同的文件系统，所以它要维护一些描述整个虚拟文件系统和真实的被装配的文件系统的数据结构。非常令人感到迷惑的是， VFS用超级块和inode节点的方式来表示系统的文件，这与 EXT2文件系统使用超级块和 inode节点的方式完全一样。与EXT2文件系统的inode节点一样， VFS的inode节点也用于描述系统中的文件、目录以及虚拟文件系统(VFS)的内容、拓扑结构。从现在开始为了避免混淆，我会使用 VFS inode节点和VFS超级块以区别于EXT2的inode节点、超级块。

在每个文件系统初始化时，它向 VFS进行注册。这个过程发生在系统启动操作系统自我初始化的过程中，真实的文件系统或者是安装在内核中的，或者是作为内核的可载入模块。文件系统模块只有在系统需要它们时才会被载入，例如 VFAT文件系统如果以内核模块的方式存在的话，它会在装配 VFAT文件系统时被载入。每当包含文件系统的块设备被装配时 (包括根文件系统)，VFS都会读入它的超级块。每种类型文件系统的超级块读例程必须要确定出整个文件系统的拓扑结构，并把这些信息映射到 VFS超级块数据结构中。VFS文件系统包含了系统中所有装配文件系统的列表和它们的 VFS超级块信息。每个VFS超级块包含执行某些特殊功能的例程的信息和指向它们的指针。例如代表装配的EXT2文件系统的超级块包含指向读 EXT2文件系统inode节点的例程。这个读例程像所有文件系统读inode节点的例程一样，会填充 VFS inode节点的对应域。每个 VFS超级块都有一个指向文件系统第一个 VFS inode节点的指针。对根文件系统来说，这个指针指向的 inode节点是用来表示“/”目录的。上面所讲的信息映射对 EXT2文件系统是非常高效的，但对其他的文件系统效率可能会降低。

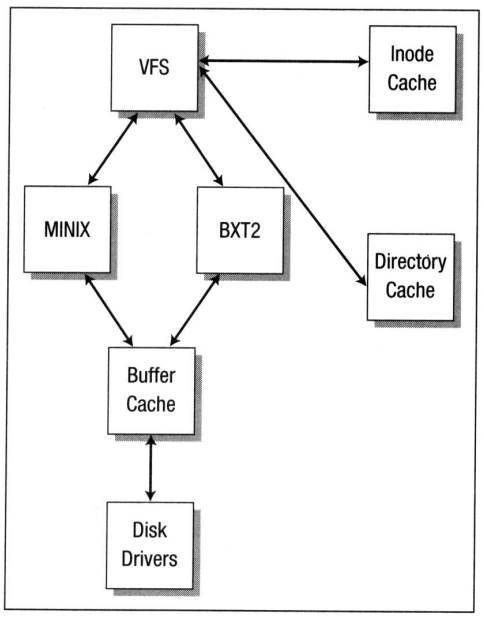

图1-7-4 VFS文件系统的逻辑结构图

在系统进程访问目录和文件时，系统会调用搜索系统中 VFS inode节点的进程。例如，对一个目录敲ls命令或对文件使用 cat命令都会使VFS文件系统搜索代表其所在文件系统的 VFS inode节点组。由于系统中的每个文件和目录都是由一个VFS inode节点表示的，所以有一些 inode节点会被经常访问。这些经常被访问的 inode节点被记录在缓存中以加速访问过程。如果要访问的 inode节点不在inode缓存中，那么系统会调用文件系统的专门例程来读入相应的 inode节点。读入inode节点的操作会使得它被加入到 inode缓存中，而对该inode节点的后续访问使得它被保存在 inode缓存中。那些访问频率较低的 VFS inode节点会被从inode缓存中删除掉。

所有的Linux文件系统使用共同的缓冲区缓存来缓存从下层物理设备来的数据，通过这种

方式来加速文件系统对它们对应的物理设备的访问。缓冲区缓区是独立于文件系统的，并被集成为Linux内核用于分配、读写数据缓冲区的一种机制。它的显著优势是使得 Linux文件系统从下层物理介质和支持物理介质的设备驱动程序中独立出来。所有的块设备向 Linux内核进行注册，并向上提供一个统一的、基于块操作的异步接口，即使像 SCSI这样复杂的块设备也支持相同的接口。在真实文件系统从下层物理硬盘读数据时，它会向控制该设备的设备驱动程序发出读物理块的请求。缓冲区缓存集成了这个块设备接口。所有由文件系统读出的块都被放入由所有的文件系统和 Linux内核共享的全局缓冲区缓存中，缓存中所有的缓冲区是由它的块号和所在设备的标识来标识的。所以，如果经常需要访问相同的数据的话，那么这些数据可以直接从缓冲区缓存中取出而不是从硬盘直接读出 (从硬盘读数据块花费的时间要长一些)。有些设备支持预读操作，这时系统推测可能使用的数据块会被设备读出，以防文件系统需要它们。

VFS文件系统还支持目录查找缓存机制，所以经常使用的目录的 inode节点可以更快地被找到。如果要做个实验的话，你可以对最近没有使用的目录作一下 ls操作。第一次ls操作时你可能会注意到一个短暂的停顿，而第二次它会立即返回结果，目录缓存存放的不是代表目录的inode节点(这些inode节点是在inode缓存中的)，而只有一些全目录名和对应索引节点的节点号的映射。

7.2.1 VFS文件系统的超级块

每个装配的文件系统由一个VFS超级块来表示， VFS超级块中主要包括下列几个域：

设备：该文件系统所在的块设备的设备标识。例如 /der/hda/——系统中第一个 IDE硬盘，它的设备标识是0X301.

inode指针： mounted inode节点指针指向文件系统中的第一个 inode节点。Covered inode节点指向代表该文件系统装配点目录的 inode节点。根文件系统的VFS超级块没有Covered指针项。

块长度：以字节为单位表示的该文件系统的块长度。如 1024字节。

超级块操作：指向该文件系统的一组超级块例程。这些例程主要由 VFS用于读写inode节点和超级块。

文件系统类型：指向装配的文件系统的 file_system_type数据结构的指针。

文件系统特征：指向该文件系统所需信息的指针。

7.2.2 VFS文件系统的inode节点

像EXT2文件系统一样， VFS文件系统的每个文件、目录等对象都是由一个 VFS inode节点表示的。每个VFS inode节点的信息都是由文件系统的专门例程从下层文件系统的信息中获得的。VFS inode节点只存在于内核的存贮空间中，只要它们对系统有用就一直被记录在 VFS inode节点的缓存中， VFS inode节点主要包括下列域：

- 设备 该VFS inode节点表示的文件系统对象所在物理设备的标识。
- inode号 inode节点的节点号，在本文件系统中是唯一的。设备和 inode号一起可以在VFS中唯一标识该VFS inode节点。
- 模式 像EXT2文件系统的这个域一样，它表示对 VFS inode节点的访问权限。

- 用户标识　拥有者标识。
- 时间　创建、更改和写的时间。
- 块长度　用字节为单位表示的本文件数据块的长度。
- inode操作　指向例程地址块的指针。这些例程是该文件系统专有的。用于操作该 inode 节点，如对该inode节点表示的文件作删减操作。
- 计数　当前使用的本VFS inode节点的系统进程数，值为0表示本inode节点可以被自由的丢弃或重用。
- 锁　本域用于锁定VFS inode节点。例如当它被文件系统读出时， VFS文件系统可以锁定它。
- 脏　表示该VFS inode节点是否被更改过。如果有改动，那么下层文件系统也要作相应的更改操作。

7.2.3 注册文件系统

在建立Linux内核时，可以选择支持的文件系统。在内核被建立后，文件系统的启动代码包含对所有安装在内核中的文件系统的初始化例程的调用。 Linux文件系统也可以作成模块的形式，它们可根据需要载入内存或用 insmod命令手工载入。

在文件系统模块被载入时，它向内核注册；在卸载时向内核注消。每个文件系统的初始化例程向VFS文件系统注册，并由一个 file_system_type数据结构来表示。该数据结构中包含文件系统的名字和指向它的 VFS超级块读例程的指针。图 1-7-5给出了在由file_systems指针指向的表中file_systm_type数据结构的格式。每个file_system_type数据结构包含下列信息：

- 超级块读例程　这个例程在文件系统的实例被装配时由 VFS文件系统调用。
- 文件系统名　该文件系统的名称：如ext2。
- 所需设备　这个文件系统需要支持设备吗？并不是所有的文件系统都要有保存它们的设备。如/proc文件系统不要求有支持它的块设备。

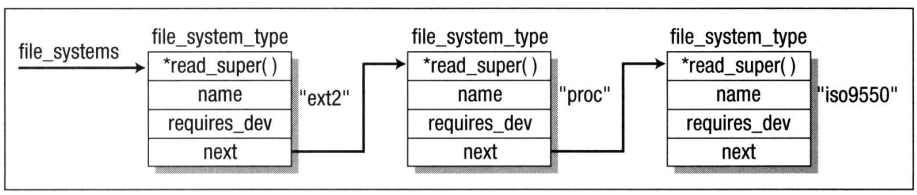

图1-7-5 注册的文件系统

通过查看/proc/tilesystems，可以看到当前注册的文件系统：

如：

```
        ext2
nodev proc
        iso9660
```

7.2.4 装配文件系统

在超级用户要装配文件系统时， Linux内核必须先验证传入系统调用中的参数的有效性。尽管mount确实做一些基本的检查，但它并不知道内核中安装了哪些文件系统以及指定的装配点是否实际存在。请看下面 mount命令的例子：

```
$ mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
```

这个mount命令会向内核传送三部分信息：文件系统名、包含该文件系统的块设备和新文件系统在当前文件系统拓扑结构中的装配位置。

VFS文件系统的第一个工作就是查找待装配的文件系统。它通过查看由 file_systems指向的表中的每个file_system_type数据结构，来搜索已知文件系统的列表。如果它找到了一个匹配名，就知道该文件系统是内核所支持的。从 file_system_type结构中，VFS文件系统可以获得读该文件系统超级块的文件系统专有例程的地址。如果它找不到匹配的文件系统，那么只要内核支持按需载入内核模块的话，一切还没有结束，这时内核在继续处理之前会要求内核守护进程载入适当的文件系统模块。

接下来如果由mount命令传送来的物理设备还没有被装配的话，VFS文件系统就必须要找到作为新文件系统的装配点的目录的 inode节点。该VFS inode节点可能在inode缓存中，也可能从装配点文件系统的物理块设备上读出。一旦 VFS找到了inode节点，就要检查该VFS inode节点是否代表着一个目录以及是否有其他的文件系统装配在这里。因为同一个目录不能作为一个以上文件系统的装配点。

这时，VFS文件系统的装配代码要分配一个 VFS超级块，并把装配信息传递给这个文件系统的超级块读例程。所有系统的 VFS超级块都被记录在super_block数据结构的super_blocks向量中，每次装配操作都必须分配一个超级块。超级块的读例程用从物理设备读取的信息填充VFS超级块的各域。对 EXT2文件系统来说，这种信息的映射或转换非常容易。它只是读入EXT2文件系统的超级块，并用它来填充 VFS文件系统的超级块。对象 MSDOS这样的文件系统就不是很容易实现了。无论什么类型的文件系统，填充 VFS超级块意味着文件系统必须从它所在的块设备中读取描述信息，如果块设备无法读取或块设备中没有这种类型的文件系统，mount命令就会失败。

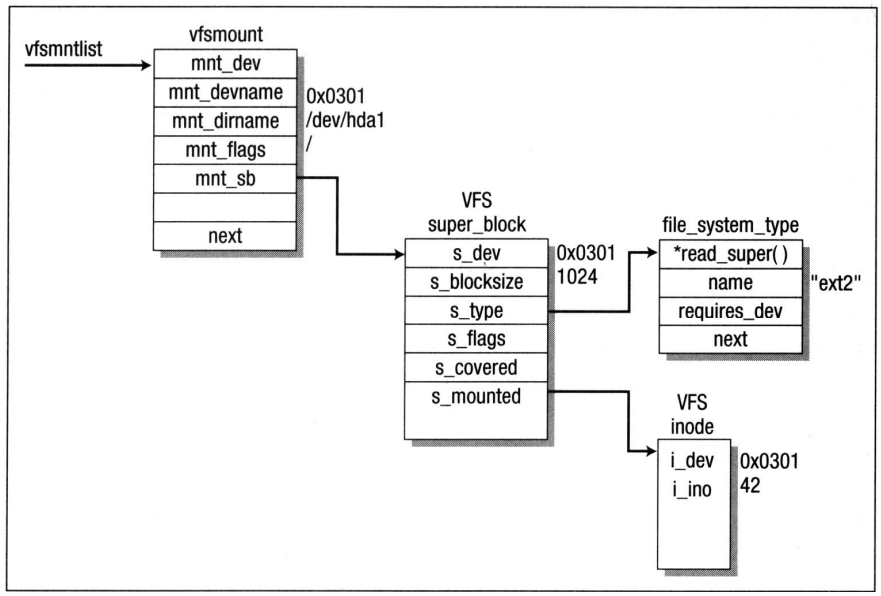

图1-7-6 安装的文件系统

每个装配的文件系统由 Vfsmount数据结构来表示。在图 1-7-6中，它们被放在由vfsmntlist

指向的队列链表中。另一个指针——vfsmntail指向表中的最后一项，mru_vfsmnt指针指向最近被使用的文件系统。每个vfsmount结构包括记录该文件系统的块设备的设备号、文件系统的装配目录和文件系统装配时分配的VFS超级块的指针。每个VFS超级块除了包含指向其对应文件系统的根inode节点的指针外，还指向它对应文件系统的file_system_type数据结构。每个文件系统的inode节点在本文件系统加载后一直驻留在VFS inode节点的缓存中。

7.2.5 在虚拟文件系统中查找文件

为了在VFS文件系统中查找某个文件的VFS inode节点，VFS文件系统必须以每次一个目录的方式来解析文件名，查找代表文件名中间目录的VFS inode节点。每个目录查找过程都包含对代表它父目录的VFS inode节点记录的文件系统专有的查找过程的调用。由于我们总是能通过该文件系统的超级块找到该文件系统的根VFS inode节点，所以上面的办法是可行的。每当真实文件系统查找一个inode节点时，系统都会先检查目录缓存。如果目录缓存中没有该目录项，那么真实文件系统可以通过下层的文件系统或者在inode缓存中找到要找的VFS inode节点。

7.2.6 卸载文件系统

如果系统的某些进程正在使用该文件系统的某个文件的话，该文件系统不能卸载。例如当系统的某个进程正在使用/mnt/cdrom目录或它的子目录的话，就无法卸载装配在/mnt/cdrom目录上的文件系统。如果有某些进程正在使用要卸载的文件系统的话，在VFS inode节点的缓存中就会有来自该文件系统的VFS inode节点。检查程序通过在inode节点表中查找来自于该文件系统所在设备的inode节点可以检测出这个问题。如果装配的文件系统的VFS超级块被标记为“脏”，这说明该超级块被改动过，所以文件系统要把它写回到硬盘上的文件系统中。一旦超级块被回写到硬盘上，由VFS超级块占用的存贮区就被归还给内核的自由存贮区池。最后为装配操作建立的vfsmount数据结构与vfsmntlist解除链接，被释放掉。

7.2.7 VFS文件系统的inode缓存

在访问装配的文件系统时，这些文件系统的inode节点被不断的读出、写入。VFS文件系统维护一个inode节点的缓存以加速对所有装配的文件系统的访问。每当有一个VFS inode节点被从inode缓存中读出时，系统就节约了一次对物理设备的访问。VFS inode节点缓存是用哈希(hash)表的形式实现的，表中的每一项是指向有相同哈希值的VFS inode节点的链表。VFS inode节点的哈希值是从它的inode节点号和包含它所在文件系统的下层物理设备标识计算出来的。当VFS文件系统要访问一个inode节点时，它先查找VFS inode节点缓存。为了找到在缓存中的inode节点，系统先计算它的哈希值，然后用这个哈希值作为inode哈希表的索引。这会返回给系统一个指向有相同哈希值的inode链表的指针。接着系统逐个读链表中的每个inode节点，直到找到了一个inode节点号和设备标识都与系统要寻找的inode节点完全相同的inode节点。

如果系统在缓存中找到了该inode节点，inode节点的引用计数加1，以表示又有另一个用户在使用该节点，然后文件系统的访问继续进行。否则的话，系统必须找到一个空闲的VFS inode节点，以便文件系统能够从硬盘上读出该inode节点。VFS文件系统在获得空闲inode节点

时可以有几种选择。如果系统可以分配多个 VFS inode节点的话，它会分配几个内核页，把它们折成新的空闲inode节点并加到inode表中。系统中所有 VFS inode节点除了在inode节点的哈希表中之外，还存在由 first_inode指向的一个链表中。如果系统已经分配了所有的可分配的inode节点，那么它必须要找到一个可重新使用的候选 inode节点。好的候选 inode节点指那些使用计数为0的节点，它表明当前系统没有在使用这些 inode节点。像文件系统的根 inode节点，这样真正重要的 VFS inode 节点，它的使用计数总是大于 0的。不可能被重新使用。一旦一个候选的重用inode节点被分配了，系统要先作清空操作。而 VFS inode节点可能被标记为“脏”，这时它需要被回写到文件系统中；或者它也有可能被锁定了，系统这时必须等到它被解锁后才能继续。候选的VFS inode节点在能重用之前必须先被清空。

在找到新的VFS inode节点之后，系统要调用文件系统的专有例程从下层真实文件系统读取信息来填充它。在该VFS inode节点被填充的同时，它的引用计数变为 1，并且被锁定以保证在它包含有效信息之前没有其他的进程可以访问它。

为了获取实际需要的VFS inode节点，文件系统需要访问几个其他的 inode节点。在你读一个目录时会发生这种现象；仅仅最后一个目录的 inode节点是我们需要的，但中间目录的 inode节点也必须被读取，随着对 VFS inode缓存的使用，它不断地被填满，最少使用的 inode节点被丢弃而使用率高的inode节点被保存在缓存中。

7.2.8 目录缓存

为了加速对经常使用的目录的访问， VFS文件系统维护着一个目录项的缓存。在真实文件系统查找目录时，它们的详细信息会被加入到目录缓存中，下次查找同一个目录时 (如列出目录中所有内容或打开该目录下的某个文件)，系统会在目录缓存中找到它。只有短目录项(最多15个字符)才会被缓存起来，但由于短目录名更经常被使用，所以这种方式是比较有道理的。如/usr/X11R6/bin在运行X服务器时经常会被访问到。

目录缓存包含一个哈希表，表中的每一项都指向有相同哈希值的目录缓存项的链表。哈希函数用该文件系统所在设备的设备号和目录名来计算对哈希表的索引值。它使得系统能较快地找到缓存的目录项。如果查找过程在缓存中花费很长时间才能确定是否会找到目录项的话，那么缓存不会带来任何好处。

为了保证缓存的更新和有效性， VFS文件系统保持最近被访问 (Least Recently Used,LRU)目录缓存项的列表。当目录项第一次被查找时，它被放在第一级 LRU表的尾端。在缓存满的情况下，它会替换掉该 LRU表前端的目录项。如果该目录项被再次访问的话，它会被提升到第二级LRU缓存表的尾端。如果缓存满的话，它会再次替换掉第二级 LRU缓存表前端的目录项。这种对第一级和第二级 LRU表前端的替换操作是符合 LRU规则的，因为位于表前端的目录项是最近没有被访问的。如果某些目录项被访问过，那么它们会更接近于表的尾部。在第二级LRU缓存表中的目录项比第一级 LRU缓存表的目录项要安全一些。它表明第二级的目录项不仅被查找过而且最近它们还被不断地访问过。

7.3 缓冲区缓存

在使用装配的文件系统时，它们会向块设备发出大量的读、写数据块的块请求。所有的读、写数据块的请求都通过标准内存例程调用以 buffer_head数据结构的形式发送给设备驱动

程序。在buffer_head数据结构中给出了块设备驱动程序所需的全部信息，有唯一标识设备的设备标识，通知驱动程序读哪一块的块号。所有的块设备都被当成相同大小块的线性集合。为了加速对物理设备的访问，Linux维护着一个块缓冲区的缓存。系统的所有块缓冲区都被放在这个缓冲区缓存中(包括新的未使用的缓存区)。这个缓存由系统的所有块设备共享，在某一时刻缓存中有很多分属于不同块设备、处于不同状态的块缓冲区。如果缓存中有系统所需的有效数据，那么它会减少一次对物理设备的访问。任何用于从块设备读出或写入数据的块缓冲区都被放入块缓冲区缓存中。随着时间的推移，一个缓存区可能因为为其他更有用的块腾出空间而被删除，也可能由于不断地被访问而一直保留在缓冲区缓存中。

缓存中的块缓冲区是由它所在设备的设备标识和对应块的块号唯一标识的，缓冲区缓存由两个功能部分组成。第一部分是自由块缓冲区的表。系统支持的不同长度的缓冲区都对应一个表。当自由块缓冲区被创建或被从缓冲区缓存中删除时，它们都会被放到这些自由块缓冲区的表队列中。系统当前支持的缓冲区大小可以是 512、1224、2048、4096、8192字节。第二个功能部分就是缓存本身，它是一个哈希表，其中每个项是指向有相同哈希索引值的缓冲区链表的指针。哈希索引值是通过该块所在设备的标识和数据块的块号来产生的。图 1-7-7给出了带有几个缓存项的哈希表，块缓冲区或者在某个自由表中或者在缓冲区缓存中。如果它们在缓冲区缓存中，就被放到 LRU表队列中。每种类型的缓冲区都对应一个 LRU表，系统用这些表执行像把缓冲区中数据回写到硬盘这样的工作。缓冲区缓存的类型反映出它的状态，Linux支持如下几种类型：

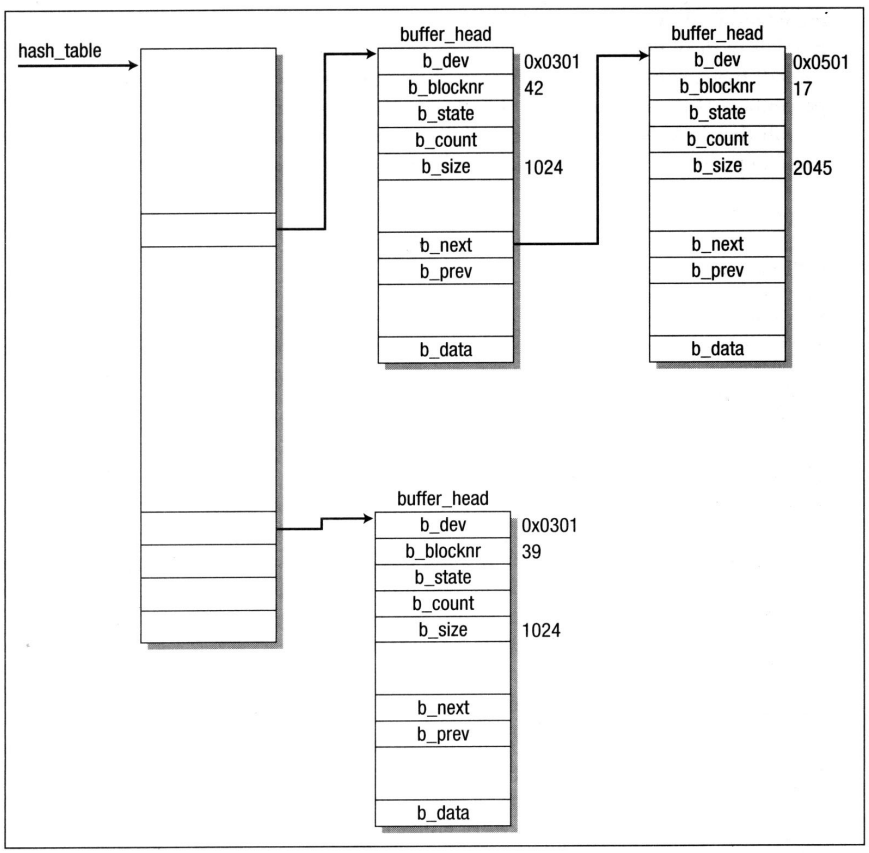

图1-7-7 缓冲区缓存

- 空白(clean) 未使用的新缓冲区。
- 锁定(locked) 缓冲区被锁定，等待写入。
- 脏(dirty) 脏缓冲区，它们包含新的将要被写回到硬盘上的有效数据，但系统还没有调度它们进行写操作。
- 共享(shared) 共享的缓冲区。
- 未共享(unshared) 缓冲区曾经被共享过，但现在没有处于共享状态。

文件系统需要从下层物理设备读缓冲区时，它会先试图在缓冲区缓存中找到该块。如果缓冲区缓存中没有，文件系统就会从相应大小的自由块表中找一个空白的缓冲区并把它加入到缓冲区缓存中，即使文件系统在缓冲区缓存中找到了这一块，该块也可能不是最新的。对新的块缓冲区和非最新的块缓冲区，文件系统要请求设备驱动程序从硬盘上读入相应的数据块。

像所有的缓存一样，系统必须维护缓冲区缓存以使得它是高效的并且对所有使用它的块设备来说能公平地分配缓存项。Linux使用bdflush内核守护进程对缓存区执行日常的维护工作而其他的工作是在使用缓存时自动进行的。

7.3.1 bdflush内核守护进程

bdflush内核守护进程是一个对有太多脏缓冲区的系统做出动态响应的简单的内核守护进程。脏缓冲区指包含还没有被回写到硬盘上的新数据的缓冲区。在系统启动时，它被作为内核的一个线程运行；但非常令人迷惑的是这时它的名字是 kflushed，也就是你使用ps命令查看系统进程时所看到的名字。这个进程大多数时间处于睡眠状态，等待系统中的脏缓冲区数目变得过大。每次分配释放缓冲区时，系统都要检查脏缓冲区的数目，如果它占系统总缓冲区数目的百分比太高的话，系统就会唤醒该进程。缺省的阀值是 60%，但如果系统急需缓冲区的话，也可以唤醒bdflush。阀值可以由update命令设置或查看：

```
# update -d

bdflush version 1.4
0:    60 Max fraction of LRU list to examine for dirty blocks
1:   500 Max number of dirty blocks to write each time bdflush activated
2:    64 Num of clean buffers to be loaded onto free list by refill_freelist
3:   256 Dirty block threshold for activating bdflush in refill_freelist
4:    15 Percentage of cache to scan for free clusters
5:  3000 Time for data buffers to age before flushing
6:   500 Time for non-data (dir, bitmap, etc) buffers to age before flushing
7:  1884 Time buffer cache load average constant
8:     2 LAV ratio (used to determine threshold for buffer fratricide).
```

当向缓冲区写入数据而把它们标志为“脏”时，所有的脏缓冲区都被链接到 BUF_DIRTY LRU表队列中；每次bdflush都会向脏缓冲区对应的硬盘中写入一定数缓冲区。这个数还可以由update命令查看和改变，它的缺省值是500。

7.3.2 update进程

update命令不仅是一个命令，也是一个守护进程。在系统初始化过程中它作为超级用户运

行，周期性地把旧的脏缓冲区写回到硬盘上。它通过调用一个与 bdflush的任务几乎相同的系统服务例程来完成上述工作。当系统使用完脏缓冲区时，它会被标记上系统时间，以便于确定何时应该被回写到硬盘上。每次update进程运行时，就会查找系统中的所有脏缓冲区，找出那些超期的脏缓冲区，把它们写回到磁盘上。

7.4 /proc文件系统

/proc文件系统才真正显示出了 Linux VFS文件系统的能量。/proc目录、子目录以及下面的文件都不是真正存在的。那么你怎么可以对 /proc/devices用cat命令呢？/proc文件系统像真实的文件系统一样向VFS文件系统注册自己。VFS文件系统在打开它的文件或目录，请求它的inode节点时，/proc文件系统利用来自内核的信息创建这些文件、目录。例如：内核的/proc/devices文件是从内核描述设备的数据结构创建来的。

/proc文件系统为用户提供了一个查看内核内部工作的只读窗口。像 Linux内核模式这样的Linux子系统，都在/proc文件系统中创建实体。

7.5 特殊设备文件

Linux像所有版本的 UNIX™一样，把它的硬件设备作为一个特殊文件看待。例如：/dev/null是空设备。设备文件不使用文件系统的任何数据空间，它只是设备驱动程序的访问点。EXT2文件系统和Linux VFS文件系统都把设备文件实现成特殊类型的 inode节点。系统有两种类型的设备文件：字符和块设备文件。在内核中设备驱动程序实现了文件的语义：你可以对它们执行打开、关闭等操作。字符设备允许为字符模式进行 I/O操作而块设备要求所有的I/O操作都经过缓冲区缓存。当设备文件收到的一个 I/O请求时，会把请求传送给系统中相应的设备驱动程序。这个设备驱动程序有时并不是真正的设备驱动程序而是像 SCSI设备驱动程序层那样的某些子系统的伪设备驱动程序。设备文件由标识设备类型的主设备号和标识主设备类型对应实例的次类型来访问。例如对系统第一个 IDE控制器的IDE硬盘，它的主设备号是 3，而IDE硬盘的第一个分区的类型号为 1。所以对/dev/hda1使用ls-l命令会产生如下结果：

```
$ brw-rw—   1 root    disk       3,    1  Nov 24  15:09 /dev/hda1
```

在内核中的每个设备都由一个两字节长的 kdev_t数据结构唯一地表示。kdev_t结构的第一字节包含次设备号，而第二字节包含主设备号。上面的 IDE设备在内核中被记为 0x0301。一个代表块或字符设备的 EXT2文件系统的inode节点，在它的第一个直接块指针中记录下设备的主设备号和次类型号。当 VFS文件系统读设备文件时，代表该设备文件的 VFS inode数据结构把自己的i_rdev域置成对应的设备标识符。